Universitat Politècnica de Catalunya

A study of neural behavioural differences among CNN architectures

Author:
Víctor Giménez Ábalos

Tutor:
Ulises Cortés García
Director:
Dario Garcia Gasulla
Co-director:
Armand Vilalta Arias
Specialisation:
Computer Science

Spring 2019

Abstract

The purpose of feature extraction on convolutional neural networks is to reuse deep representations learnt for a pre-trained model to solve a new, potentially unrelated problem. This process would enable the use of deep learning for tasks with small data size and for people without the skills or access to the computational capabilities required for training nets.However, raw feature extraction from all layers makes training most ML methods unfeasible given the massive size of these networks. As such, extra processes must be performed to facilitate it. In this work, we study, reimplement, evaluate and improve an already existing method for doing so: the Full-Network Embedding. We evaluate our work with empirical tests on real datasets, and in comparisons with the previous state-of-the-art in several metrics, such as the accuracy and execution time. We manage to improve the execution time for the full pipeline and acquire further comprehension on the stochasticity of the method to the point it simplified through a mathematical formula. We also discover different alternatives that can substitute part of this pipeline, based in similar approaches with different granularity, supervised and unsupervised mechanisms and more, achieving accuracies comparable to the state-of-the-art of the original FNE. In the process, we discover properties of neural behaviour in CNNs which may be exploited both for transfer learning and deep learning.

Keywords: transfer learning, machine learning, neural networks, convolutional neural networks, feature extraction, deep learning.

Resumen

La extracción de activaciones neuronales de una red neuronal convolucional permite reutilizar representaciones internas de un modelo pre-entrenado para resolver problemas nuevos, incluso no relacionados con el problema original. Este proceso abre el uso de aprendizaje profundo a problemas préviamente no considerados por falta de datos, así como a personas cuyos recursos computacionales y/o capacidades no les permitían entrenar redes neuronales. Sin embargo, la mayoría de métodos de aprendizaje máquina no pueden entrenarse con la salida de una extracción de todas las capas de la red dado el tamaño de dichas redes. Para ello, se deben realizar procesos adicionales. A lo largo de este trabajo, estudiamos, reimplementamos, evaluamos y mejoramos un método ya existente: el Full-Network Embedding. Basamos nuestros experimentos en resultados empíricos y datasets reales y en comparativas contra el estado del arte previo en métricas como precisión y tiempo de ejecución. Conseguimos mejorar el tiempo de ejecución del proceso completo y simplificar el componente estocástico del método mediante una fórmula matemática. Además, descubrimos alternativas que permiten sustituir partes del proceso. Algunos los basamos en el mismo enfoque que el estado del arte pero con grano diferente, mientras que otros siguen diferentes filosofías como métodos supervisados contra no supervisados. En el proceso, descubrimos propiedades de comportamiento neuronal en redes convolucionales, que pueden ser explotados tanto para el campo de aprendizaje por transferencia como para aprendizaje profundo.

Palabras clave: aprendizaje por transferencia, aprendizaje máquina, redes neuronales, redes neuronales convolucionales, extracción de activaciones neuronales, aprendizaje profundo.

Contents

1 Introduction to the problem at hand

In the field of AI, a problem that is being faced continuously is the image classification problem, in which we try to classify an image into a class. This is the case in many industrial (*e.g.*, qualitycontrol, autonomous driving, predictive asset management, targeted advertisement) and medical (*e.g.*, image diagnosis) tasks. The current state of the art are *Convolutional Neural Networks*[1]. These are a form *Neural Networks* specifically tailored to deal with multidimensional inputs, which makes it suitable for the task of image classification.

However a degree of further explanation is required before we delve into these. A *Neural Network* is a machine learning algorithm inspired on the way neurons interconnect in the human brain, albeit quite simplified. In its most basic form, a neuron is a computational unit which has a vector of weights w. This neuron takes an input vector x, performs an inner vector product (in this case, sum of the element-wise multiplication) between the input and weights w. The result is a single number, which optionally can be treated by an 'activation function' g. Formally: $output = g(\sum_i^n x_i * w_i)$. We might also refer to this 'neuron' as a 'feature', and to the output value as a *neuron activation* or *feature value* in the embedding space.

In the context of neural nets, a layer is a set of these neurons, each taking the same input, multiplying by their different sets of weights and outputting a new vector of length equal to the number of neurons. Stacking several layers, feeding one into the next, is a neural net. In the case of a multinomial classification task, the most popular application of neural networks is to have the last layer contain as many neurons as the number of classes we are dealing with. By applying the *softmax* activation function, we can understand the net's output as the 'probability' of the image being of each class (so output one would be associated to class one and subsequently).

The magic of neural networks is that, by taking inputs and getting the resulting deliberation of the net (the output of the last layer), the net can be 'corrected' based on what we would want it to output. When doing so, the weights of this net are changed based on mathematical formulae to better 'fit' the result. Put in a more formal way, we are in front of an optimisation problem in which we want to minimise our error by modifying the set of weights. In the multinomial classification problem, the error is typically computed through the categorical-cross entropy. This process is performed through *backpropagation*, or propagating the error backwards through the net, and using the gradient of the loss to modify the weights.

In a *Convolutional Neural Networks*, or CNN, the neuron is modified by taking into consideration the nature of spatiotemporal correlation in some data, inspired by the biological neurons in the optical nerve. In data modalities with locality properties (such as images or series of chronological events), datapoints near-by are more relevant than far-away. As such, sparsifying the neurons so they process small and nearby datapoints reduces the complexity of training, as well as overfitting to the training data. This is implemented by making some neurons share weights, and have each one consider a subset of the input. Each of these shared-weight neurons process different subsets. For example, taking the first three numbers of the vector gives one result, and then taking the elements two to four gives another result (see Figure 1). This allows an extraction of big amounts of information whilst keeping the number of weights to be optimised reduced and conserving the locality of the activation.

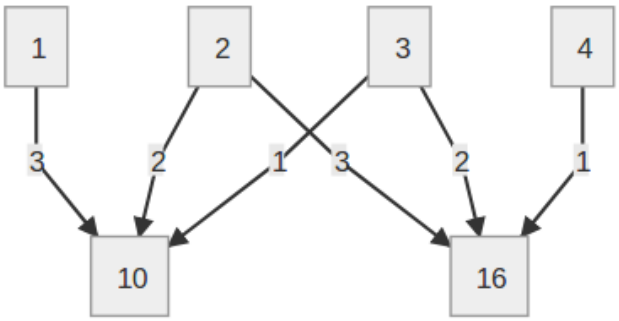


Figure 1: Example of a convolution with input {1,2,3,4}, weights {3,2,1} and output {10,16}.

This is particularly interesting in the field of image classification, since we can process a small chunk of the image (three by three pixels) and a neuron could be interpreted as 'is this pattern present at this position of the image', where the pattern is defined by the weights of the neuron. A CNN is simply a form of neural net in which there exist layers of this kind[2]. Stacking convolutional layers results in the model being able to recognise more complex patterns: first layer neurons may detect vertical lines, middle layer neurons may detect noses or eyes, and deep layers neurons may recognise faces.

We take notice some of the problems of this technique (especially for deep nets with many neurons):

- **Computational power:** The optimisation step is computationally costly, requiring expensive hardware and even accelerators. And even with them, training is time consuming.
- **Labelled data requirements:** In order for this technique to work, we need a very large set of labelled images for the task we want to classify. Small data size does not enable the model to 'generalise' for new, unseen images.
- **Expertise:** The specification of a net requires a researcher who is apt at designing the net, tailoring it to the specific task at hand through hyper-parameter tuning: the number of layers, the number of neurons in each layer, the activation functions, number of convolutional and fully connected layers...
- **Learning results are opaque:** After it has learnt, understanding the content of the net is difficult (each weight is a number and understanding the relationships between them is hard) and we do not know what each neuron is doing.

The AI community is extremely interested in solving the second of these, data availability. For some domains such as medical images it is quite difficult to obtain a dataset of the required size. If we were to somehow remove this requirement, CNNs could be applied to several domains that were previously intractable.

Through dispelling the opaqueness of what is learnt we might be able to solve the problem of data availability. We do this by noticing how a CNN may take as input images from domains completely different from what it was trained in. Whereas the output from such an experiment

might not seem relevant, the excited neurons' output are a function of the image presented. The internal representations learnt for the original problem may be used to *describe* the new image, even though they were not trained for it. Formally, the probability of any image of belonging to a class may be conditioned by the neural activations of a network trained for one specific task, and vice-versa. Exploiting this relationship, training the neural network is no longer necessary, and with it we make the problem available to methods with lower data amount requirements.

We will do so by *forward-propagating* (presenting) images from our problem through a pre-trained CNN, extracting neural activations (internal representations), and exploiting these through other machine learning techniques for classification. This process is one of the applications of the field of *Transfer Learning*.

As an example, we could present images of flowers to a CNN trained in recognising dog breeds, and using the internal representations extracted from the net to ascertain what kind of flower we had.

1.1 Stakeholders

- **Developer:** As the one who researches and writes both code and documentation, he is the one in charge of dealing with the deadlines and problems in general.
- **Members of the HPAI group:** The group that the developer belongs to and the people behind the basic idea of this research, the contribution may result in prestige and potential funding from future projects. In particular we refer to the speaker, the director and the co-director.
- **Researchers using AI:** Our results in these experiments may contribute to a methodology for applying AI in domains of image classification where it was unfeasible. In particular, researchers of the medical field may benefit greatly given the scarcity of labelled medical data. Furthermore, it may provide researchers with altogether new and better tools for traditional image classification problems with a high number of classes.

However, the stakeholders may not be limited to the aforementioned. The width of the impact is dependant on the future applications derived from our results. Potentially, these could be applied to domains not mentioned nor considered.

1.2 State of the Art

The use of Transfer Learning techniques for problems has been studied in several contributions [3, 4, 5, 6]. In most cases, the authors tried to extract information from or fine-tune a pre-trained CNN. Given the size of the model, they focused on the later layers, presumably the most informative. An example of this is "CNN features off-the-shelf: an astounding baseline for recognition"[3], where focus is on the extracted activations from the first fully connected layer (the first non-convolutional one). In "How transferable are features in deep neural networks?"[4] the analysis is focused on preserving the beginning of the net, locked or unlocked for training, and retraining later layers from scratch. In "Factors of transferability for a generic convnet

representation"[5], this idea is expanded to study the effectiveness related to the similarity of the tasks (the one used to train the model and the problem at hand).

In this work, however, we do not restrict ourselves to the last layers. We base our analysis on two contributions [7, 8] which consider the whole net's output, and further study the methodology proposed.

"On the Behavior of Convolutional Nets for Feature Extraction"[7] presents a method to extract the discriminativity of a neuron (also referred to as feature) of the net with respect to a class of the new problem. The authors talk of discriminativity as the information the value of the activation gives us with respect to the possible class of the image. Plainly put, it establishes a method for saying whether neural response is representative, irrelevant or unrepresentative with respect to a class. We might also refer to this relationship as a neuron being positively discriminative, non-discriminative or negatively discriminative.

"An Out-of-the-box Full-network Embedding for Convolutional Neural Networks"[8] follows up with a method to exploit this discriminativity to establish a discretised Full Network Embedding (vector of information from all neurons), which is highly informative and can be used as the input of other methods achieving state-of-the-art results for some tasks. In particular, the paper empirically finds two thresholds for discretising the value of a neuron into -1, 0 or +1, and this is used by another classifier. They used a Support Vector Machine with Linear Kernel and hyperparameter C=1.

From these two contributions, a procedure is established for the purpose of performing transfer learning with a CNN net with good results.

However, in these there are several points which require further study:

- **The transformation of the FNE may not be optimal.** A global threshold disregards neuron characteristics, such as layer type, depth in the net and activation distribution. These may yield valuable information.
- **The study for building FNE defines two thresholds achieved empirically from a net architecture on three datasets.** Consistency of these thresholds across different datasets is unknown.
- **The computational cost of analysis is expensive.** In order to perform these explorations, cluster level hardware is required. This makes parameter exploration and fine-tuning of the method difficult for private entities. If the procedure could be changed or optimised it may allow for better results from the method, or open the field for other researchers to work with this and explore it themselves, hastening the research on the method.

2 Scope

Our main objective in this work is to further analyse the FNE methodology explained in both of the aforementioned contributions [7, 8], in an attempt to improve its different aspects and to make its implementation feasible. To do so, we defined a series of objectives and revised them as we discovered new potential improvements.

- **Improving the execution time of the method:** The necessary hardware to perform the exploration on this problem is quite demanding: with big problems a computational cluster is needed. This makes analysis of the method difficult for private entities. By the application of Amdahl's law, we find the most expensive parts of the method and optimise them to drastically reduce the execution time.
- **Revising discriminant threshold obtention:** We put special focus on the thresholding methodology used in the discriminant space. The process for obtaining these is one of the most computationally costly. Furthermore, it is stochastic. By finding a way to predict these thresholds, we remove a substantial amount of execution time, and we deepen our understanding of the underlying mathematical processes in the method.
- **Revising the feature thresholding methodology used:** Discretisation of the FNE is necessary given the high number of neurons in the model, whose output are real numbers. Traditional classification algorithms would not be able to learn otherwise due to the curse of dimensionality [9]. The final thresholds proposed are a constant generalised from three experiments, in three different datasets. The procedure to obtain said thresholds is stochastic, depends on the target task, and disregards the difference between neurons within the net. We revise this methodology in order to see if there is a margin of improvement.

3 Methodology and validation

In order to make clean, understandable code, ensure correct results, maintain a quick development and evaluate the quality of our solutions, we employed the following strategies, tools and protocols:

- **Test Driven Development:** We used this methodology in the beginning stages of the code, to ensure quick development and minimal debugging. As the experimentation and variations began, we stopped using them: the design was in a state of flux and these tests took more to code than the implementation itself. However, by this point the end-to-end was built and evaluation mechanisms took its place. These are described below.
- **Git Issues:** The tool provides an ever-present global picture of what our current objectives were and what our progress in each of them was, while also leaving room for change and expansion.
- **Meetings with the directors:** Daily meetings with the director and co-director route the problem if the findings did not match what we expected.
- **Comparison with legacy data:** While a simple comparison is not possible given the stochasticity of some of the steps and the fact that we work on different frameworks (Python 2 vs Python 3), by always keeping an eye out for strange divergence from original experiments we were able to detect problems as early as possible in the development cycle.
- **Comparative studies with legacy code and third party methods:** To ensure efficient code, while cutting on development time cost, we find the most time-consuming snippets of code and optimise them as much as possible and outperform generic library calls.

In addition, we used the following metrics for evaluating our success:

- **Validation by regression error:** We find that there is a possibility to predict the outcome of heavy code-snippets. To ensure this result is valid, we will use Machine Learning techniques, such as Leave-one-out cross-validation with to ensure that our results are indeed as good as they seem.
- **Validation by classification results:** In the end, as one of the final objectives of this project is achieving a model for classification, we can test our hypothesis against a classifier and compare our accuracy with what we expect from previous data.
- **Validation by explainability:** Some of the parts of our codes attempt to delve into the complexities of a neural network and its behaviour, as well as some statistical tests. We will try to explain this behaviour through analysis, as plausible explanations reinforce the validity of our results.

4 Planning

Preliminar background acquisition started on October 2019. The project development lasted five months, from January 28th, 2019, to June 26th.

Next, we detail each of the tasks performed during the project.

4.1 Task description

4.1.1 Acquiring background in Neural Networks, Deep Learning, Transfer Learning and others

In order for the work to progress, the developer must have a theoretical background on many topics, such as Neural Networks, Deep Learning, Transfer Learning, statistics, mathematics and many others. For that purpose, the developer read papers on these subjects, as well as attended to courses on them (such as MAI's course on Deep Learning, datacamp's courses on Deep Learning. Tensorflow's Crash Course, and UPC's Introduction to Statistical Learning Theory lessons). In particular, the developer had to read all the aforementioned papers on the current state-of-the-art and familiarise with the methodologies used in each of them.

For that, a computer as well as stationery were needed as material resources. Human resources (the developer) were also necessary to understand the documents and courses.

This process started before the project, on October 2018, and is expected to be revisited along the course to keep the developer updated and to contrast results obtained.

4.1.2 Study on legacy code, data formats

Access to legacy code has been granted to the developer in order for him to work on. The formatting of the data and the mechanisms of the previous code needed to be studied and

understood in depth to allow for improvements and extensions. In the case that part of this code is not understood, the developer met with the previous developer to clear doubts, thus potentially avoiding scheduling delays.

The needed resources were a computer, stationery and human resources.

4.1.3 Coding main features and optimisation

The developer decided that, in order for the code to be extended, it would be more cost effective to start over from zero, as it was an experimental code never intended for production and unfeasible to maintain. Coding the main features the previous code allowed for a better understanding of the previous code and for optimisations from the start. In addition, this work includes the conversion of legacy data into a new format to work with the new code.

These results were validated by comparison to legacy data. The results were equal, and execution time was reduced substantially in the process.

This task was a bottleneck of the project, as the rest of tasks were all dependant on this one.

In the material resources department, aside from the previous, access to computational clusters such as the Nord3 and Minotauro from BSC was granted. Moreover, access to the datasets used on previous studies, or at the very least the activations was also required. Finally, we also require the *Python* programming language interpreter and a set of libraries, such as tensorflow, caffee, numpy, etc.

4.1.4 Study over variability and stochasticity of the method

Using the framework made in the previous task, the developer ran experiments to study the stochasticity and variability of the results, which previous work did not do.

In this process, we found that the variability of the method is indeed bounded, as explained in Section 9.2.1. Along with this finding, we found that the inter-dataset variability can be explained. This created the new task below.

The resources used are the same as in the previous task, and its time allocation has remained the same.

4.1.5 Study over the inter-dataset threshold variability

From the previous task, we found that the thresholds may be predicted with little information about the target task. We attempted to do so via *Machine Learning* and statistical methods. We decided to allocate much more time to this task, making it a cornerstone of the project, and its time allocation heavily modified the initial planning of the project.

We justify this decision with the objectives defined. Predicting the discriminant threshold eliminates a heavy overhead from the pipeline. Furthermore, it gives insight into how the method works.

4.1.6 Study of the prediction error

We performed a study of the error performed when using the approximate method of the previous task. We care for the error in the value of the threshold, but also in what it implies in the whole method.

Particularly, we were interested in seeing how many feature-class pairs change, as that is what directly reflects in our results. This is to validate our results of the previous task.

4.1.7 Study over noise distribution

From the *study of inter-dataset threshold variability* task, we deduce that in order for this prediction to work, there must be something that is predictable in the pipeline. We thought it was related to the noise in the discriminant space.

Once again, this goes in line with the previous task's justification: it will improve execution time and increase our insight in the method, potentially changing it for the better.

4.1.8 Feature threshold experimentation

We will put focus on the feature thresholding mechanisms, which are key to the method's accuracy. We considered expanding the possibilities to improve the final accuracy and offer insight on the nature of extracted features.

4.1.9 Document project

Writing documentation both for the technical and social part of the project. Each document needs to show the work in a comprehensive and didactic approach, as well as leave no detail out to allow for reproductibility of our results. This task was performed during all of the project to ensure good documentation for expanding the project. The time allocated to this task will be much higher than in other projects due to the paper submission. We allocated a total time of 120 hours.

4.2 Task time allocation

Table 1: Revised time allocation table to each of the tasks

Task	Time(h)
Acquiring background	75
Study legacy	40
Coding main features	90
Stochastic variability	50
Inter-dataset variability	80
Error study	20
Discriminant noise distribution	45
FT experimentation	100
Documentation	120

4.3 Gantt Diagram

In Figure 2 we present the Gantt diagram of the project. Note that both background acquisition and documentation expand throughout the whole project, but are allocated the same amount of time as detailed in the previous section.

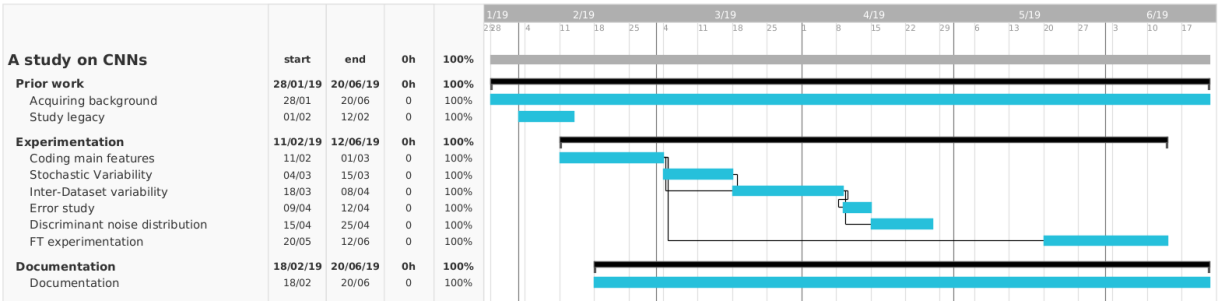


Figure 2: Gantt diagram of the project

5 Budget

5.1 Project budget and costs

In the following sections we detail the different costs associated with the project's development, as mentioned in previous sections.

The project's final budget was the predicted in the second report (May 31).

5.1.1 Hardware

We account for the cost of the laptop, as well as the cluster's amortisation based on its original cost and the execution time expected. The specific clusters that we used were the *MinoTauro* and the *Nord3*. *Nord3*, is a subset of the previous *Marenostrum 3* decomissioned in 2018. This technically means that it has survived its useful life and amortisation period. *MinoTauro*'s cost is not public but we can give a rough estimate of the overall devaluation cost based on general information of the *BSC* supercomputers.

Marenostrum 4 costed €34 million, and both Marenostrum 2 and 3 lasted around 6 years, which are around 52560 hours. Dividing these costs gives us 650€/h. We expect our code to be running in the machines for 55 hours approximately, however we also need to factor that our code is not the only thing running concurrently in the cluster. Moreover, we are working with only one of the racks of the original Marenostrum 3, which in a rough estimate should factor as merely a fraction of it. Specifically, a rough estimate based on the amount of computation nodes and racks, should be 1/90th, which brings the total cost to 400€.

The amortisation cost is computed as the fraction of time of the useful time that the product will be in use for the project (in our case, the laptop will be used for 5 months out of the 8 years of useful life).

Table 2: Hardware costs

Product	Cost	Useful life	Amortisation
MSI GP62 2QE Leopard Pro	1499.99€	8 years	78€
Cluster usage estimate	-	-	400€
Total	1499.99€		478€

5.1.2 Software

All software used is free of cost, therefore the total software cost as well as the amortisation is zero.

Table 3: Software costs

Product	Cost	Amortisation
LaTeX	0€	0€
Ubuntu OS	0€	0€
Python and libraries	0€	0€
Tensorflow	0€	0€
Git	0€	0€
Sublime Text	0€	0€
TeamGantt	0€	0€
Total	0€	0€

5.1.3 Human resources

The project was developed integrally by a single person acting in different roles. However, these roles have different salaries associated.

Table 4: Human resources cost

Position	€/hour	hours	Total
Project Manager	45€	30	1350€
Software developer	20€	320	6400€
Data analyst	50€	100	5000€
Total		450	12750€

Regarding these costs, the project manager was in charge of ensuring that deadlines were met, the project did not derail and what time allocation was given to each of the tasks (as seen in its variability in previous sections). He wrote the less technical documentation, such as the logs and milestones achieved and the state of the project.

The software developer was in charge of all the coding-related tasks, as well as preparing the experimentation frameworks. Moreover, he was also had to familiarise with previous work and background on the subject, particularly in the coding strategies, optimisation techniques and knowledge of the programming language. In addition, he was responsible for writing the code documentation.

The data analyst was in charge of taking decisions related to project focus and experiment design. He took technical decisions of statistical or machine-learning nature. Furthermore, all the documentation related to decision making was written by him.

5.1.4 Unexpected costs

In case of unexpected events, an extra time allocation has been set for all the positions to account for the possible additional costs.

Table 5: Unexpected costs

Position	€/hour	hours	Total
Project Manager	45€	5	225€
Software developer	20€	50	1000€
Data analyst	50€	20	1000€
Total		75	2225€

We estimated the number of extra hours from the tasks performed and the previously detailed contingency plans, the manager being the less involved in possible unexpected developments, and the data analyst being in charge of steering the project in case of unexpected experimental results.

5.1.5 Indirect costs

Regarding the indirect costs, the stationery cost was given an ample upper bound. The electricity was computed with the cost of 0.12€/kWh in mind (which is the mean cost of electricity in Spain, data extracted from Endesa).

To estimate the power usage of the laptop, referring to specific model, we took 22 W as the mean power consumed in active mode, and 10 W as the mean power consumed during power saving mode. We estimated that 80% of the time, the laptop works in active mode, which gave us an average consumption of 18.8W. From this, and knowing we allocated approximately 500 hours of work to the project (all of them using the laptop), we estimated a cost of 9.4 kWh, or 1.13 €.

In order to compute the consumption of the clusters, we used the internal tools provided by the cluster to know how much energy a job took. From some of starting jobs (which make for a good sample to estimate the real costs) we generalised that an hour of computation of our jobs took 0.164 kWh. Multiplying this by the approximate number of hours of code run in the cluster (55 hours) and the cost of electricity associated to our jobs gave us the cost estimate of close to 1.08€.

The internet connectivity was computed from the monthly cost of 35€, which will be used for 5 months (not taking into account previous exploratory work of the background acquisition task).

Table 6: Indirect costs

Source	Cost
Stationery	50 €
Electricity (laptop)	1 €
Electricity (clusters)	1 €
Internet connection	175 €
Total	227 €

5.1.6 Budget summary

Table 7: Budget summary

Source	Cost
Hardware	478 €
Software	0 €
Human Resources	12750 €
Unexpected costs	2225 €
Indirect costs	227 €
Total	15680 €

Furthermore, we also tried to map the costs to the actual tasks of our previous report.

Table 8: Task budget

Task	Cost(€)
Acquiring background	1692
Study legacy	902
Coding main features	2500
Stochastic variability	1128
Inter-dataset variability	1805
Error study	451
Discriminant noise distribution	1002
FT experimentation	3000
Documentation	3200
Total	15680 €

5.2 Budget control

In order to control the budget, the manager compared the real costs with the integrated costs at the end of each task, taking into account the difference between the estimated and real costs of each task. With these formulae, we detected budget deviations and were able to correct them.

Cost Deviation = $(EC - RC) * RH$

Efficiency Deviation = $(EH - RH) * EC$

where E and R refer to estimated and real, and C and H refer to cost and hours, respectively.

As seen from the initial budget allocation, most of the costs ended up coming from the human resources.

We therefore created an 'unexpected costs' category, which estimates the extra costs based on the extra amount of hours each of the positions might have had to put into the project. This ended up happening, and as such we drained some of the budget allocated to this task. In addition, to compensate for this, some of the original tasks were re-scheduled or removed from the project

6 Sustainability

We show and discuss the sustainability matrix to evaluate the project's sustainability, and analyse each of its parts: environmental, economical and social.

6.1 Dimensions of sustainability

6.1.1 Environmental aspect

As seen in previous sections, the cost of the electricity consumed in the code execution is negligible. The ecological footprint of the personal hardware used is estimated as $320kgCO_2$, without taking into account the possibility of recycling (which would bring it down to $290kgCO_2$). This

data is extracted from a study from Dell applied to the laptop.

Even though the previous work is being reused (as it would be inefficient for our purposes), all of the material resources of the project, as well as the final product, will be reused in future projects. The framework provided by the main features of the project can be reused for further analysis beyond the scope of the project and the hardware is still be useful by its end. We cannot do much about the indirect costs, although we expect to cover them with the results provided.

Moreover, with this study we hope to lower the computation times and requirements for state-of-the-art techniques on transfer learning, which will have an arguably positive repercussion on the environment during its useful life. Transfer learning by itself drastically reduces the computations needed to deal with many machine learning problems, as it removes the need to train a neural net from scratch. Precisely the project will reduce the ecological footprint of solving these problems.

Finally, we consider this project to have no environmental risk whatsoever. The results of this project have no application which could be harmful, but many in which it would be helpful, such as reducing the cost of machine learning in general.

6.1.2 Economical aspect

As seen in previous sections, the costs associated with the development of the project have been computed and are deemed at an appropriate level.

Furthermore, the costs of performing these computations with respect to the previous state-of-the-art has gone down, as it is expected that the new improvements on the code will allow it to run in less expensive hardware (rather than on a supercomputer). This is not reflected in the PPP costs, but rather in the subsequent usage of the results of the project.

However, the project has the risk of not becoming popular or that other alternatives become mainstream. It is possible that the development team does not recovering the project's investment.

6.1.3 Social aspect

This project provided the author himself with a deep background in neural networks as well as knowledge on the discipline of *Transfer Learning*. In addition, the task of documentation and background acquisition by themselves will prove as a positive experience for future projects and research, as well as provide a reusable template with which the author is already familiar.

Regarding the social repercussions, we expect the project to push the state-of-the-art in areas where neural networks are hard to apply, such as medical data. The expansion of AI to this field will be positive for society as a whole, and that is only one of its possible applications.

In addition, the removal of neural network training facilitates solving problems for people without resources, allowing working on small problems which were previously cost-ineffective. The usage of neural networks to deal with problems will no longer be restricted to big corporations, universities and/or people with access to clusters. By democratising deep learning we make it accessible to people with less powerful machines, and the reduction in both costs and data requirements will open previously disregarded problems.

Given the possibilities that the success of this work could bring, the author considers this project to be useful to society and that there is a real need for the it.

Furthermore, great part of the project is staked on the reproductibility of its results, as we remained transparent in our procedures. This will benefit researchers confirming and expanding the research of the topic.

In contrast, the possibility of this project impacting society negatively is brought from the very same reasons. A democratisation of deep learning might mean giving tools to private business to exploit data in an unethical way, when previously it was too costly to be done. In the author's opinion, the opportunities outweigh the risks but this does not mean they are non-existing.

6.2 Sustainability matrix

Below are the scores assigned to each of the aspects of sustainability.

Table 9: Sustainability matrix

	PPP	Useful life	Risks
Environmental	Design consumption	Ecological footprint	Environmental risks
	9/10	15/20	0/-20
Economical	Bill	Viability plan	Economical risks
	8/10	15/20	-5/-20
Social	Personal impact	Social impact	Social risks
	9/10	18/20	-3/-20
Sustainability Range	26/30	48/60	-8/-60
	68/90		

7 Law and regulations

The only parts of the project that intersect with possible law issues are the usage and results from the datasets in use. As of now, all of the datasets used are public, open source and/or under the Creative Commons license. The libraries and programming tools used are also open source.

The legacy code used as a reference through the work is available on github under an open source license, thus no legal problem may come from that aspect.

8 Experiments

8.1 Algorithm explanation

To enter into the project's specific objectives, a further explanation of the method proposed in previous work [7, 8] is required. We want to solve a classification problem of images, which we will refer to as the *target task*. We attempt to do so by transforming the images into something

which can be understood by traditional machine learning methods. We do so by using a CNN model, the *source model*, trained in another problem, the *source task*.

In previous work, a method was established to obtain an *embedding*, or representation of the images to be treated. This representation is a vector in a smaller dimension space than the pixel level information of the original image. Its features encode relevant information while disregarding local information.

First, to obtain more training data as well as improve our accuracy with a voting mechanism, the train and test images are upsampled taking 10 crops out of each one (four corners, centre and mirror). These images are forward propagated through the trained CNN model, collecting the net's activations. We extract the activations after the activation function (ReLU). A spatial average pooling is then performed over the convolutional layer outputs, to reduce the dimensionality. All the image activations collected are standardised feature-wise, so that for each feature the mean activation is 0 and its standard deviation is 1. We obtain this standard deviation and mean from the train set only, but we use them to normalise the test too.

This by itself could be a good *embedding*, although it's high complexity coming from a highly dimensional continuous space may entail problems related to the curse of dimensionality. Discretising this embedding solves these problems. After this discretisation, the data is ready for training an SVM, and the previous crop upsample can be used to implement a voting classifier, improving the accuracy. This part of the process can be seen in Figure 3

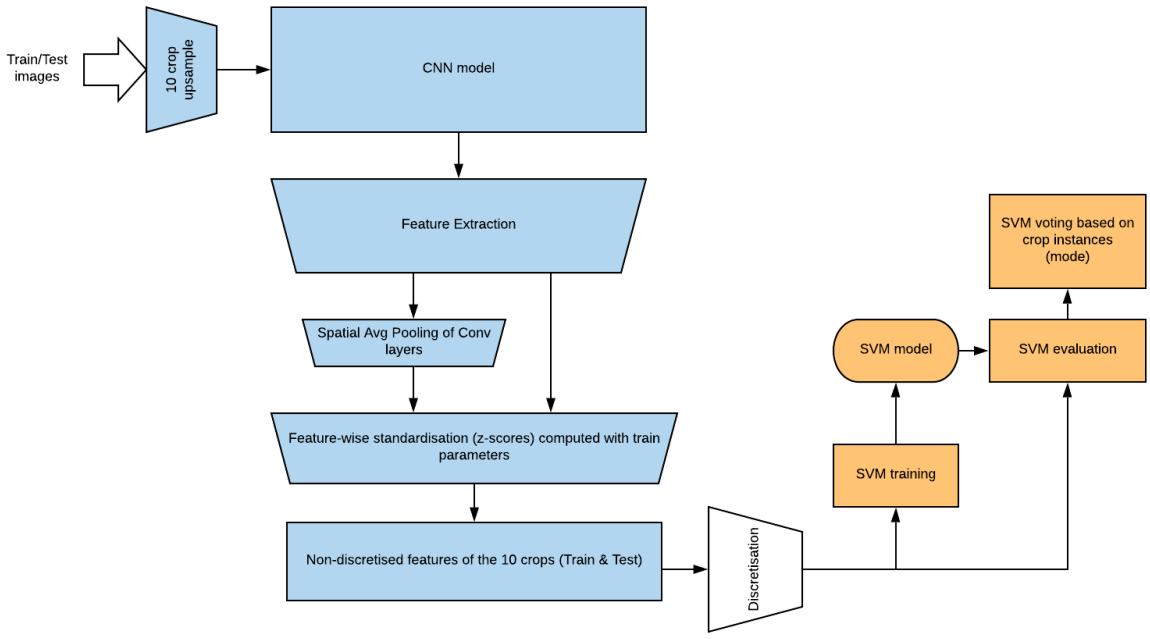


Figure 3: Full Network Embedding process, from initial dataset to SVM model. Blue is the preprocessing steps up until discretisation. Orange is the training and evaluation of the SVM.

However, the discretisation is yet to be explained. We want to obtain values in the activation space such that we can consider an activation significant if it exceeds these values[8]. We explain the process in the following paragraphs, and it is also illustrated in Figure 4. In addition, to

ensure the correctness of the method, we undo the crop upsampling by taking the average of the 10 embeddings as a representative for the image.

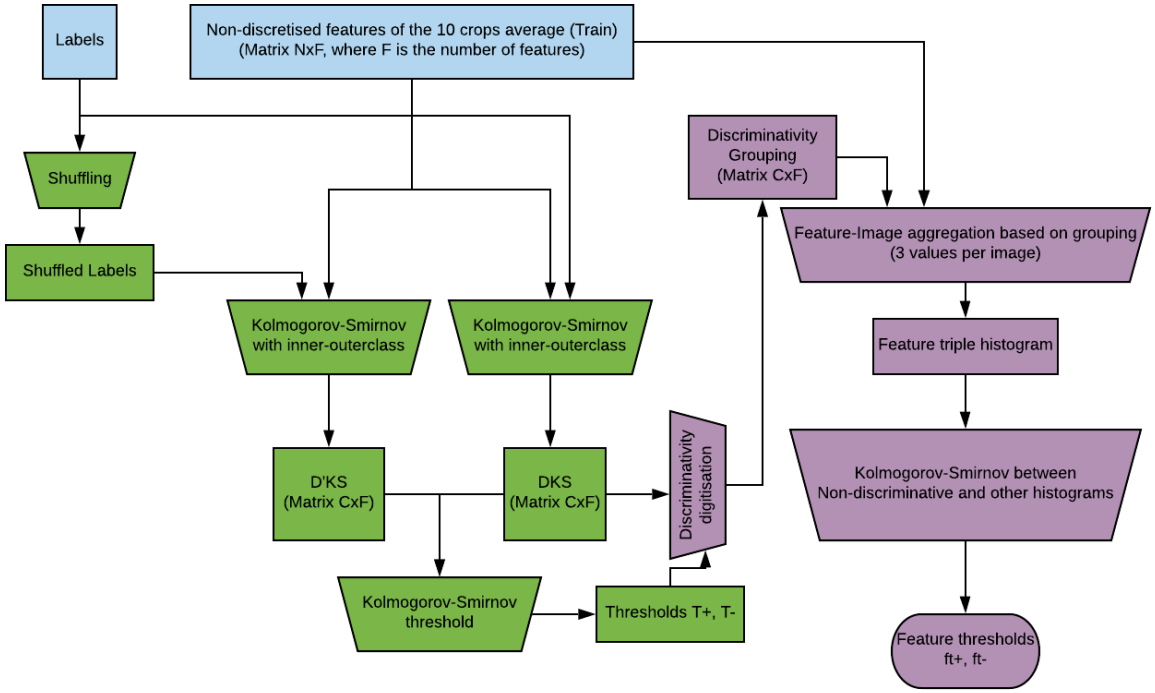


Figure 4: Described process to obtain the feature thresholds. In green are the steps for obtaining the discriminant thresholds t^+, t^-. In purple, the steps for obtaining the feature thresholds ft^+, ft^-

To obtain the values for discretisation, which we name ft^+, ft^-, we consider the discriminativity of a feature toward a class: a feature is positively discriminative if high activation values point toward the stimulus image being of that class. This is similarly considered for negatively discriminative, and neutral. This **grouping of feature-class pairs** into three discriminant groups allows us to take the activations holistically in three groups to obtain at which values is it more likely that the activation is discriminating for or against classes in general. This can be done by calculating the point at which the distributions are furthest away from one another, as explained below. These thresholds proved to perform well on a variety of tasks in previous work.

To obtain such a grouping of feature-class pairs, one needs to determine their discriminativity. The proposed methodology groups all the *embeddings* of the target task by the class of the original images. For each feature in the embedding, it computes the signed Kolmogorov-Smirnov distance between all the feature's activations in each class against the rest. This yields a value between -1 and 1 indicating the feature's discriminativity toward a certain class. We will refer to this value $D_{KS}(f, c)$, where f is the feature and c is the class; this is the discriminativity of a feature-class pair. In Figure 5 is an example of the KS calculation. The blue distribution are a feature's activations on stimulus of images not of a class, and in green the same but for images of a certain class. This feature-class pair is $D_{KS}(f, c) > 0$.

The required grouping will be formed through discretising these discriminativity measures with

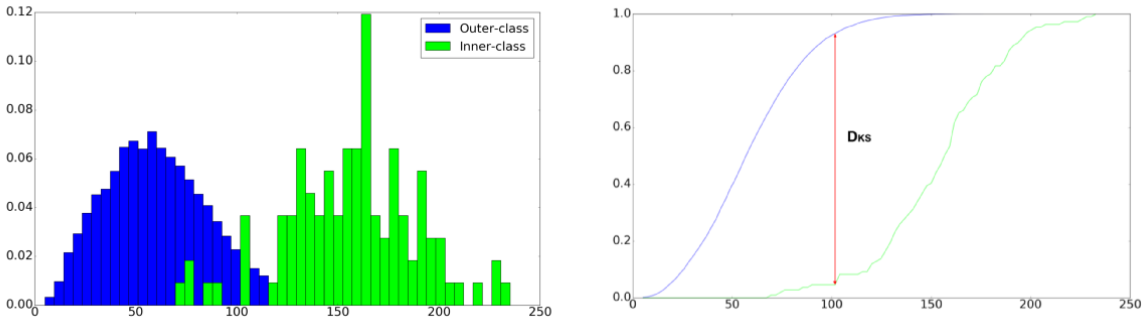


Figure 5: Kolmogorov-Smirnov between inner and outer class activations, extracted from [7].

two thresholds, one for positive and one for negative. We will name these thresholds t^+ and t^-, respectively, and we will find them by estimating the level of noise in the discriminativity.

To do so, the authors in the previous work[7] proposed to repeat the process after shuffling the labels of the data randomly. The obtained discriminativity values are named $D'_{KS}(f,c)$, and intuitively they should all give values of random discriminativity (as now inner and outer class are random images). Therefore, this is mostly noisy discriminativity.

Now, symmetrically for positive and negative D_{KS}, the authors calculate the accumulated distributions of the real and the shuffled distances toward 0. Formally, for the positive side,

$$A^+(t) = \sum_{f \in F} \sum_{c \in C} I_{[t,+\infty)}(D_{KS}(f,c)),$$

where F is the set of features in the embedding, C are the classes in the target task and $I_J(x)$ is 1 if x is in interval J and 0 otherwise. This is done for the D'_{KS} too: $A'^+(t)$. This procedure yields four distributions (real and shuffled for positive and negative).

To find a trade-off between noisy and real discriminativity, the authors computed the point at which A and A' are further away from one another using Kolmogorov-Smirnov:

$$\underset{t}{argmax}(A(t) - A'(t)).$$

These values obtained are in the D_{KS} space, one for positive (t^+) and one for negative (t^-).

One may then form the grouping $G(f,c) \in \{-1,0,1\}$, where the value depends on whether $D_{KS}(f,c)$ is below, between, or above the t^+ and t^- thresholds.

To discretise the feature space, the authors proposed the following mechanism. For all instances in our training set, compute the mean value of the features of the embedding depending on the group the features fall in with respect to the image's class. As a result, three values are obtained for each of the images (one for each category of grouping). An example of an histogram of these values is exposed in Figure 6. By computing Kolmogorov-Smirnov between the activations in the non-informative grouping ($G(f,c) = 0$) and the other two, we obtain the two final ft thresholds which we can use to discretise the embedding space.

Through this long process, and through obtaining three sets of these ft thresholds on three datasets, the authors decided to set them as constants $ft^- = -0.25$ and $ft^+0.15$. These thresh-

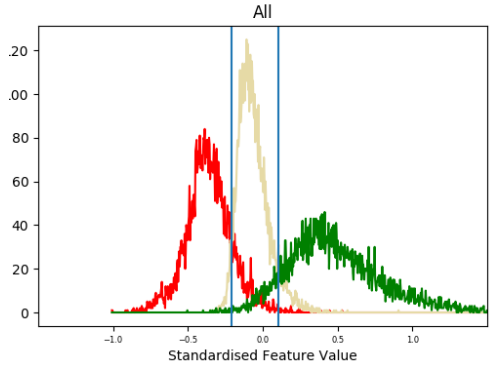


Figure 6: Example of the histograms of mean activation per group. Each histogram has as many elements as the number of images in the dataset. Red, grey and green correspond to values -1,0,+1 of the grouping. The two blue lines correspond to the resulting Kolmogorov-Smirnov values, or ft thresholds

olds are proposed as universal and independent of the problem at hand.

8.2 Method innovation

In our project we will be dealing with improvements of the method explained above. We want the following:

1. **Improving the execution time of the method**
2. **Revising the obtention of discriminant thresholds**
3. **Revising the obtention of feature thresholds**

8.2.1 Code acceleration of Kolmogorov-Smirnov calculus

This improvement is integrated within the *coding main features* task. It addresses the first of our objectives by improving the execution time of an expensive code snippet.

The calculus of Kolmogorov-Smirnov distances (especially for obtaining the D_{KS} and D'_{KS}) is a major bottleneck of our process, we put focus on improving the code with respect to the legacy version.

The legacy code discarded the use of library implementations of Kolmogorov-Smirnov, as the cost associated to them is $O(nlogn)$. This comes from the fact that to compute this metric it is necessary to compute the accumulated distribution, which requires sorting. However, given that precision is not paramount, this part of the process may be removed by using a binning approach (computing histograms), which in turn removes the need of the ordering and makes the computation be $O(n)$.

Even then, the process for computing Kolmogorov-Smirnov in our case marks a bottleneck of this process. This is because we need to compute this metric for every feature in the source model

and category in the target task, making it $O(n * F * C)$. However, by taking the fact that it must be computed for all categories, we may drastically reduce the computation time associated at the cost of extra memory usage proportional to the required precision.

The modification performed is to avoid the repeated histogram calculus for inner and outer-class. Instead of for each class and feature computing the two histograms, for each of the features we compute a histogram for each of the classes. By adding all but one of the histograms together (bin-wise), we obtain the two required histograms. We therefore reduce the call to the histogram function by half. Furthermore, instead of adding the histograms for each of the classes, we may add all of them together and substract (bin-wise) the histogram of the class we want to compute. This also enables the full usage of parallelisation mechanisms in the CPU.

We note that as a consequence of this last action, there might exist a minor decrease in numeric precision of the order of the machine's epsilon (10^{-20}). However, given that our method is already an approximate one which only considers three decimals at most, this is negligible.

In addition, by switching the data structure used to save the results we further achieve a slightly better performance.

Next we expose the code for both mechanisms.

This is the code previously used for computing Kolmogorov-Smirnov:

```
#data_matrix stores the normalised activations,
#  each row corresponds to an image, each column
#  to a feature.
#labels stores the labels of the images of the
#   dataset, in the same order as data_matrix
#results are left in the dictionary kol_div,
#  which has 1 entry per label consisting of
#  each feature discriminative toward the class

kol_div = {}
for label in np.unique(labels):
    kol_div[label] = np.zeros(data_matrix.shape[1])
    intra_data = data_matrix[labels == label]
    inter_data = data_matrix[labels != label]
    for i in range(data_matrix.shape[1]):
        kol_div[label][i] = ksd(intra_data[:, i], inter_data[:, i])

def ksd(p,q):
    nbins=100
    both = np.concatenate((p, q))
    _,bins = np.histogram(both,nbins)
    b_p,_ = np.histogram(p,bins)
    b_q,_ = np.histogram(q,bins)
    cb_p = np.cumsum(b_p).astype(float)
    cb_p = cb_p/cb_p[-1]
    cb_q = np.cumsum(b_q).astype(float)
    cb_q = cb_q/cb_q[-1]
```

```
    pq_diff = cb_q-cb_p
    abs_max = np.argmax(np.absolute(pq_diff))
    return pq_diff[abs_max]
```

This is the new code used for Kolmogorov-Smirnov:

```
#data_matrix is like in previous code
#labels is like in previous code
#results are in matrix kol_div, with each row
#  being a class, each column a feature and each
#  value the feature-class pair discriminativity.

kol_div = np.zeros((len(np.unique(labels)), data_matrix.shape[1]))
features = kol_div.shape[1]
classes = kol_div.shape[0]
for feature in range(features):
    _, bins = np.histogram(data_matrix[:,feature], 100)
    l_ind=0
    cumsums=np.zeros((classes, len(bins)-1)
    for label in np.unique(labels):
        intra_data = data_matrix[labels == label, feature]
        hist = np.histogram(intra_data, bins)[0]
        cumsums[l_ind]=np.cumsum(hist).astype(float)
        l_ind+=1
    cumsumAll = np.sum(cumsums, axis=0)
    for c in range(classes):
        kol_div[c,feature]= cumKS(cumsums[c], cumsumAll-cumsums[c])
return kol_div

def cumKS(one, All):
    one = one/one[-1]
    All = All/All[-1]
    dif = All-one
    return dif[np.argmax(np.absolute(dif))]
```

8.2.2 Machine Learning strategies to deal with threshold acquisition

In previous work, the authors detected a correlation between the threshold and the average number of instances per class (hereafter referred to as $\hat{I}_c$). We believe that through further analysis it might be faster to predict the threshold value instead of performing the whole computational process associated to it. We will attempt to do so via a Machine Learning technique: residual sum of squares (RSS) minimisation, or regression.

The authors attempted a linear regression over $\hat{I}_c$, obtaining a high R^2. Analysis of the methodology hints to the presence of a horizontal asymptote as $\hat{I}_c$ increases. For this reason, we discard the use of a linear regression. At the same time, the absolute value of the thresholds seems to be inversely correlated to $\hat{I}_c$. In preliminary studies we considered the following alternatives:

the logarithmic, reciprocal and logarithmic reciprocal. The results obtained by the logarithmic reciprocal are remarkably better than the alternatives, which is why these are the only results we show and discuss in the rest of this work. To evaluate the goodness-of-fit we calculate the R^2 of a *leave-one-out cross-validation*. The function that performed best, with a difference of more than 10% of R^2 was the logarithmic reciprocal, formally written in Function 1.

$$t(\hat{I}_c; a, b) = a + b/ln(\hat{I}_c) \tag{1}$$

In the new method, we will simply fit the values a, b of Function 1 from empirical values obtained via the method described in Section 8.1. We justify the use *leave-one-out cross-validation*. Obtaining datapoints requires computing the t thresholds on a dataset, making the obtention of a massive amount of data unfeasible. Given the size of our data to predict from, we choose *leave-one-out cross-validation*, which while being computationally expensive it is more reliable when working with small data size.

To prove the method works, we ensure a series of properties of the methodology:

1. The shuffling method proposed must have resulting threshold randomness bounded. We are interested in testing the stochasticity of the method over different problems, a factor that was not studied in [7]. If the randomness is not bounded, attempting to predict will require more tuning. In this experiment, and due to its computational expense, we limit ourselves to a subset of the datasets, which we pick to be representative. To study the randomness, we will compute the D_{KS} as explained in subsection 8.1. Then, we will compute D'_{KS} for 21 random permutations of the labels (as they are computationally expensive to obtain). With these, we obtain 21 thresholds. The magnitude of the std of the threshold's distribution will give us a measure of their randomness.

2. The $\hat{I}_c$ seems to have a very high relevance for the threshold value. In order to study this, we will perform the regression over threshold values extracted with the method detailed in section 8.1, from a single balanced dataset with varying $\hat{I}_c$. To do so, we will take a balanced dataset and cripple it to different $\hat{I}_c$, in increments of 10, and then we will fit the function's parameters, and evaluate the goodness-of-fit.

3. The regression must be able to generalise to several target datasets. To do so, we perform the regression over thresholds obtained from a variety of datasets, detailed in the following subsection. Furthermore, to ensure the method's applicability to several network topologies and source tasks, we will do this experiment for a set of source models. Once again, we will obtain the thresholds via the method detailed in section 8.1, then fit the regression and evaluate the goodness-of-fit.

4. The regressed values must not drastically increase the changes of feature-class pair type (informative or noisy). We will obtain the percentage of changes for one of the regressions, as it is important for evaluating the real error of the new methodology.

8.2.3 Studying noise distribution

The results from these previous experiments makes us suspect the high predictability must be explainable. We remember that the t thresholds come from calculating the Kolmogorov-Smirnov

distance between two distributions of D_{KS}, one with labels shuffled.

As we expect the non-shuffled D_{KS} to be informative (meaning they are relevant to the target task's characteristics beyond $\hat{I}_c$), the predictable part should mostly come from the random noise. We explore the nature of the D'_{KS} distribution among all of our dataset partitions.

8.2.4 Studying FT threshold values

We focus now on the ft thresholds, which discretise the neuron's activation space. In previous work[7], these values were computed from the average value of features belonging to a group G for each image. We refer to this original methodology as *Global*. Moreover, after performing these experiments with *mit67, cub200* and *flowers102*, they decide to set feature thresholds $ft^+ = 0.15$ and $ft^+ = -0.25$ as constants, applicable to any target task disregarding everything else. We will refer to this thresholding methodology as *Constant*, and we note that, unlike the previous one, this can be considered unsupervised (as we are not using the labels for setting thresholds).

We also note that the feature activations of each neuron are quite different. Even though these activation values have been standardised, F_f (the distribution of feature f activations) are not necessarily identically distributed. An example of this is shown in Figure 7. This hints to the necessity of treating each feature individually, or at least to not treat equally two features that behave so differently. For that purpose, we propose to find individualised $ft(f)$ thresholds, tailored to each individual feature of our source model. We refer to this methodology as *Feature-wise*.

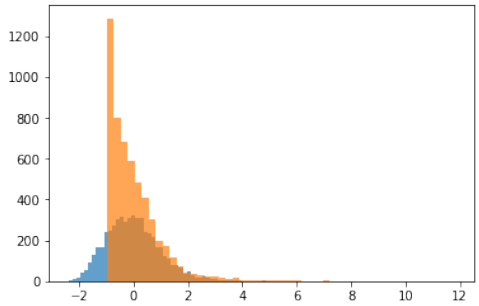


Figure 7: Histograms of activations for features 0 and 1 of *VGG16IN* on dataset *mit67TR*. Both of these features are from the first convolutional layer.

To cover the behaviour of feature behaviour in all grain levels, we study the distribution of feature thresholds *Layer-wise* (meaning studying each of the network's layers individually). This is done to detect neural behavioural differences among layers and the impact of layer depth in the ft.

In addition, for all of these methodologies, we consider only the mean activations per group for each image. We analyse the behaviour of not doing so, and instead using this methodologies taking all of the activations in their corresponding group. We refer to these two variants as mean and no-mean alternatives, and we study them for the *Global*, *Feature-wise* and *Layer-wise* methodologies.

We also try a naive approach to the problem. From the previous information about the F_f, we will assume individual feature thresholds are positive for the problem. In addition, we want to

find thresholds that maximise the information that is given to the SVM. Naively, by discretising in a way that there are exactly one third of instances in each of the groups (meaning each feature will have the same number of 1, 0 and -1), we will be diversifying the features passed onto the SVM, potentially giving the maximum amount of discretised information to the SVM. We will name this method *Quantiles*, as we will take the 0.33 and 0.67 quantiles of a feature as thresholds.

We will study the behaviour of these methodologies, along with the mean vs no mean variants. Finally, we test the methodologies which made sense with a *linear support vector machine classifier*. The process performed is the detailed previously in Section 8.1 and Figure 3. We summarise the steps next:

1. Ten-crop upsampling of all the data: we take the corners and the center of the image, as well as these images' mirror, obtaining 10 crops per original image.
2. Forward propagation of the crops through the neural network, and extraction.
3. Spatial average pooling of the convolutional layers (obtaining only one activation per neuron instead of the whole convoluted matrix).
4. Feature-wise normalisation (z-scores) of the data. Train normalisation is computed as is, test normalisation is computed with the train's parameters so that they are scaled in the same way.
5. Discretisation of the data according to methodology. Thresholds obtained from data (all but *Constant* methodology) use train data exclusively, but are applied to both train and test.
6. Training of a Linear SVM with train data, with hyper-parameter C = 1 as default. It could be optimised, but one of the objectives of the methodology is to present a method with little optimisable hyperparameters.
7. Prediction of test data with trained SVM model. The prediction uses the ten crops, obtains the labels from the classification and takes the mode of the 10 crops to classify the original image.

8.3 Experimental data

8.3.1 Source models

For the CNN architectures we use the VGG16 and VGG19 topologies[10]. These are composed by consecutive blocks of convolution and pooling layers (16 and 19 layers respectively), and two fully connected layers. This sort of architecture is quite representative of the CNN designs being used today. As for the source tasks, we use the following: *ImageNet 2012*[11], a dataset for classification spanning 1000 categories of objects, and *Places 2*[12], a scene recognition task unrelated to *ImageNet 2012* with less categories. Of the possible combinations of architecture-source task, the only case we do not have available is the VGG19 trained on *Places2*. The rest are referenced as follows: VGG16 CNN trained on *ImageNet 2012* (*VGG16IN*), VGG19 CNN trained on *ImageNet 2012* (*VGG19IN*), and VGG16 CNN trained on *Places2* (*VGG16P2*).

8.3.2 Target datasets

Since we want to obtain a generalisable method, we need to use different target tasks, ideally with different $\hat{I}_c$) and spanning different domains. We consider the following 9 datasets, freely available online:

- The *MIT Indoor Scene Recognition* dataset [13] (*mit67*) is a dataset for classification in 67 different categories of indoor scenes. Its classes depend on global spatial properties and on the relative presence of objects.
- The *Caltech-UCSD Birds-200-2011* dataset [14] (*cub200*) is a dataset containing images of 200 different species of birds.
- The *Oxford Flower* dataset [15] (*flowers102*) is a dataset consisting of 102 flower categories.
- The *Oxford-IIIT-Pet* dataset [16] (*catsdogs*) is a dataset covering 37 different breeds of cats and dogs.
- The *Stanford Dogs* dataset [17] (*stanforddogs*) contains images from the 120 breeds of dogs found in *ImageNet*. The dataset is complicated by little inter-class variation, and large intra-class and background variation.
- The *Caltech 101* dataset [18] (*caltech101*) is a classical dataset of 101 object categories containing clean images with low level of occlusion.
- The *Caltech 256* dataset [19] (*caltech256*) is similar to *Caltech 101*, but contains over the double amount of categories and minimum number of images in any category is higher.
- The *Food-101* dataset [20] (*food101*) is a large dataset of 101 food categories. Test labels are reliable but train images are noisy (*e.g.*, occasionally mislabelled), and for this reason we will only use the test set in our experiments.
- The *Describable Textures* Dataset [21] (*textures*) is a database of textures categorised according to a list of 47 terms inspired from human perception.
- The *Oulu Knots* dataset [22] (*wood*) contains knot images from spruce wood. This dataset is considered to be challenging even for human experts.

To increase the number of target tasks feeding our regression, while providing variance in $\hat{I}_c$, in some cases we consider the different data splits originally provided as different tasks. In particular, we use training sets (TR), test sets (TE), joined training and test sets (TRTE) and validation sets (VAL). From now on, all references to target tasks will regard to a specific dataset and split. The properties of the 21 resulting target tasks are shown in Table 10. Notice the *caltech101TRTE* (followed by *caltech256TRTE*) has a remarkably larger imbalance in the number of instances per class than the rest of tasks.

Table 10: Properties of all tasks used in our experiments, including average number of instances per class ($\hat{I}_c$) and the corresponding standard deviation (Imbalance).

Dataset	#Images	#Classes	Average #Images/Class	Imbalance (label σ)
caltech101TRTE	9145	102	90	123.07
caltech256TRTE	30607	257	119	85.69
catsdogsTR/TE/TRTE	3669/3680/7349	37	99/99/199	1.5/1.5/3.0
cub200TR/TE/TRTE	5994/5794/11788	200	30/29/59	0.17/2.91/2.91
flowers102TR/VAL/TE	1020/1020/6149	102	10/10/60	0/0/44
food101TE	25250	101	250	0
mit67TR/TE/TRTE	5360/1340/6700	67	80/20/100	1.39/1.39/0
stanforddogsTR/TE	12000/8580	120	100/72	0/23.12
texturesTR/VAL/TE	1880/1880/1880	47	40/40/40	0/0/0
woodTR	438	7	62	50.84

9 Results

9.1 Code acceleration of Kolmogorov-Smirnov calculus

The following tests were done in an MSI laptop, with a Intel® Core™ i7 5950HQ / 5700HQ processor and 16 GB of memory.

We measure the execution time of computing the D_{KS} values with both algorithms. This experiment is done on the extracted activations of *mit67TE*, just after the spatial average pooling. This task amounts to 13400 crops, and 12416 features per crop. The execution time for the two methods is shown in Table 11. The speed-up achieved by our code improvements (computed as the quotient between both execution times) is of 4.6.

Table 11: Execution time

Method	Execution time (seconds)	Execution time (minutes)
Old	1213.18	20.22
Optimised	262.96	4.38

This speed-up is obviously not related to the model and task aside from the size of data these entail (the number of neurons of the source model, images/crops of the target task), aside from the number of classes in the target task which is the main source of recomputations in the previous code. We limit our experiments to this task and model, given the computational cost of running the old code and considering the experiment generalisable to the rest of our cases.

Table 12: Threshold statistics for 4 datasets, ordered by average instances per class descending

Dataset	t^- Avg	$t^-\sigma$	t^+ Avg	$t^+\sigma$	Avg #Instances per class $\pm\sigma$
mit67TRTE	-0.109	0.00125	0.119	0.00050	100
caltech101	-0.140	0.00077	0.160	0.00030	89.66 ± 123.07
cub200TR	-0.174	0.00112	0.195	0.00090	29.97 ± 0.17
flowers102TR	-0.284	0.00234	0.321	0.00238	10

9.2 Machine Learning strategies to deal with threshold acquisition

9.2.1 Randomness of threshold analysis

As previously discussed, we need to assess the stability of the shuffling methodology, since we will be using it to validate the consistency of our regression model. Due to computational constraints, we only use a subset of target tasks for this purpose: *caltech101TRTE*, *mit67TRTE*, *cub200TR*, and *flowers102TR*. This subset spans different topics, have different $\hat{I}_c$, and different imbalance levels. We use the pre-trained model *VGG16IN* to obtain the corresponding random $D'_{KS}(f, c)$.

Table 12 shows the threshold values and their standard deviation for each of the selected tasks. Notice all standard deviations are at least 2 orders of magnitude below the thresholds. This fact speaks for the consistency of the stochastic methodology.

9.2.2 Regressions for discriminative thresholds

Figure 8 corresponds to the regression performed on the subsets of *mit67TRTE*. We observe a surprisingly high R^2 coefficient for both positive and negative thresholds. This supports our claim that the thresholds are predictable from the I_c.

In Figure 9 we expose the difference in behaviour caused by altering the properties of pre-trained models. In Figure 9a, we compare *VGG16IN* and *VGG16P2*, which have different source tasks. In Figure 9b we compare *VGG16IN* and *VGG19IN*, which have different architectures. In both plots of Figure 9 we observe a consistent set of outliers that are not as well adjusted as the others. Remarkably, these correspond to tasks with significant class imbalance (standard deviation above 20, as seen in Table 10). For clarity, these data points have been marked with x in the previous figures. Figure 10a is a regression on *VGG16IN* having removed these tasks: *stanforddogsTE*, *woodTR*, *flowers102TE*, *caltech101TRTE*, *caltech256TRTE*. We refer to this as the *balanced regression*. Figure 10b shows the previously fitted balanced regression, on top of the threshold values from the subsets of *mit67TRTE*. Its R^2 is computed directly with no LOOCV, as we are not recomputing the regression, but evaluating whether the previous one fits the data.

The R^2 values of all these regressions are presented in Table 13.

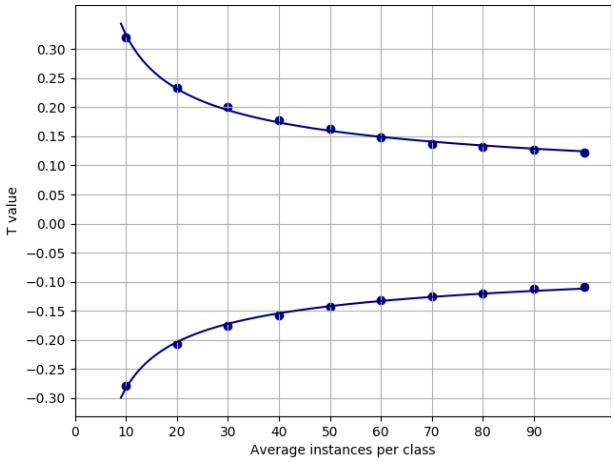
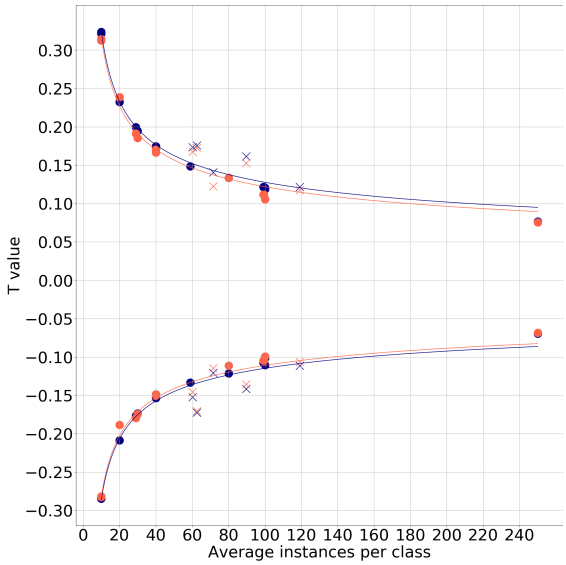


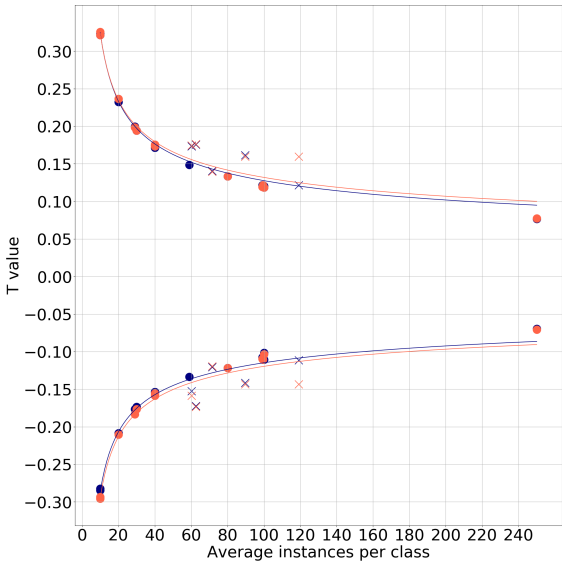
Figure 8: Regression of thresholds for subsets of mit67, with different number of instances per class. The dots correspond to empirical threshold values.

Table 13: R^2 values of each regression. *bal.* stands for the balanced regression.

Experiment	t^-	t^+
mit67 subsets	0.986	0.990
VGG16IN	0.944	0.962
VGG19IN	0.920	0.935
VGG16P2	0.936	0.950
VGG16IN *bal.*	0.995	0.997
mit67 *bal.*	0.993	0.995

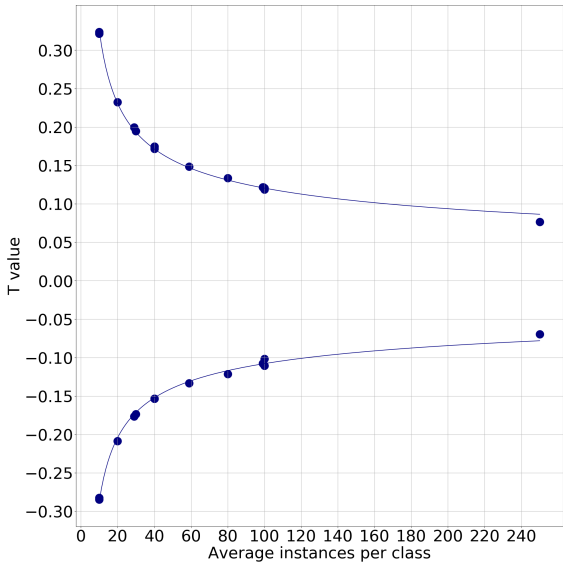


(a) *VGG16IN* (blue) versus *VGG16P2* (orange).

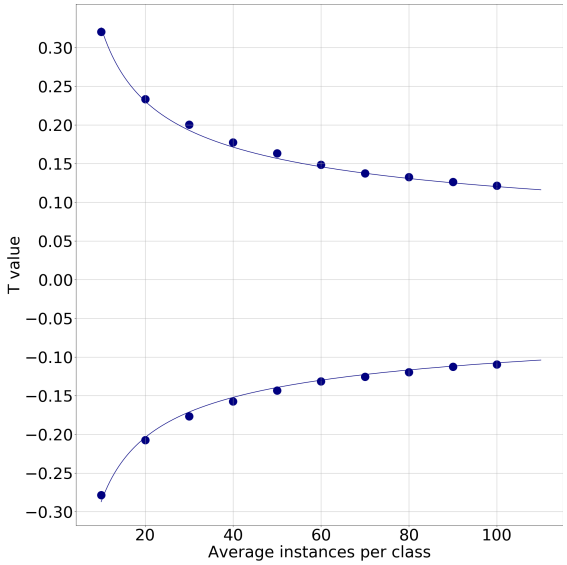


(b) *VGG16IN* (blue) versus *VGG19IN* (orange).

Figure 9: Regression over all target tasks and different pre-trained models. Marked with X are the empirical threshold values of task partitions with σ label distribution above 20.



(a) Regression with dots corresponding to empirical threshold values of the target tasks.



(b) Left function but with dots corresponding to *mit67TRTE* cut to different I_c

Figure 10: Balanced regression.

9.2.3 Imbalance error and influence

To evaluate the impact of our methodology, we perform a study on the difference between the original thresholds for *VGG16IN* (obtained with the stochastic method) and the predicted thresholds (obtained with the regression on *VGG16IN* with no filtering). In Table 14 we record the threshold values as well as the percentage of changes. Coherently, the ones with higher amount of changes are the imbalanced tasks, as well as the *food101TE* (this particular case is discussed in Section 10.1).

9.2.4 Studying noise distribution

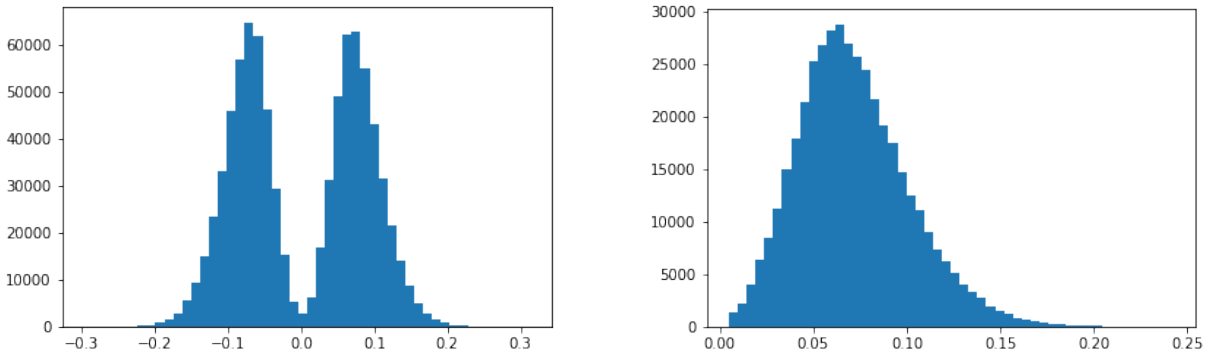
We begin by attempting an analytical observation of the $D'_{KS}(f, c)$. Each feature f follows an activation distribution F_f, which are not necessarily identically distributed (as seen in Figure 7). Each class c defines a partition of the activations in this distribution. In our particular case it is a random partition of a bigger outer-class set and a smaller inner-class set. The proportion of images in each is related to the number of classes and the imbalance of the dataset. The D'_{KS} values are simply the results from computing the Kolmogorov-Smirnov distance between both sets in these partitions.

Note that Kolmogorov-Smirnov between two tests is simply the maximum signed distance between two cumulative distributions, which in our case come from the same F_f. In Figure 11a we observe a histogram of the D'_{KS}, particularly for the *mit67TR* dataset and *VGG16IN* model, and in Figure 11b we present only one of its sides (positive).

We observe that, even though the origin distribution F_f is the same for both of the partition sets, the D_{KS} values are pushed away from zero due to the supremum of the Kolmogorov-Smirnov

Table 14: Threshold influence

Task	t^- original	t^- predicted	t^+ original	t^+ predicted	Percentage of group changes
caltech101TRTE	-0.1415	-0.1182	0.1615	0.1317	7.691
caltech256TRTE	-0.1115	-0.1077	0.1215	0.1196	1.016
catsdogsTR	-0.1085	-0.1142	0.1215	0.1271	2.105
catsdogsTE	-0.1075	-0.1143	0.1215	0.1272	2.320
catsdogsTRTE	-0.0825	-0.0915	0.0925	0.1011	3.371
cub200TR	-0.1735	-0.1753	0.1945	0.1972	0.720
cub200TE	-0.1765	-0.1776	0.1995	0.1999	0.276
cub200TRTE	-0.1335	-0.1364	0.1485	0.1526	1.241
flowers102TR	-0.2825	-0.2870	0.3215	0.3254	0.867
flowers102VAL	-0.2845	-0.2870	0.3235	0.3254	0.439
flowers102TE	-0.1525	-0.1353	0.1735	0.1514	5.503
food101TE	-0.0695	-0.0853	0.0765	0.0939	8.668
mit67TR	-0.1215	-0.1228	0.1335	0.1370	0.848
mit67TE	-0.2085	-0.2069	0.2325	0.2335	0.445
mit67TRTE	-0.1105	-0.1140	0.1205	0.1269	1.850
stanforddogsTR	-0.1015	-0.1140	0.1185	0.1269	4.553
stanforddogsTE	-0.1205	-0.1276	0.1405	0.1425	2.098
texturesTR	-0.1535	-0.1570	0.1745	0.1762	0.969
texturesVAL	-0.1535	-0.1570	0.1715	0.1762	1.473
texturesTE	-0.1535	-0.1570	0.1745	0.1762	0.957
woodTR	-0.1725	-0.1336	0.1755	0.1494	10.025



(a) Positive and negative

(b) Positive

Figure 11: Histogram of *mit67TR* $D_{KS}(f, c)$

formula. From this example we observe the particular distribution of each of the sides of this distribution. It appears to be a skewed Gaussian or a non-central Chi-square.

Following this, we try different approaches to transform this into a normal, both using logarithmic and square-root transformations, the later being the closest to being a gaussian. The histogram of the square root values of this distribution is presented in Figure 12. However, raw normality tests fail to ensure the normality of our values, possibly due to the difference in behaviour in the left-side tail. In addition, we note that the high amount of D'_{KS} values (412714), coupled with the approximate Kolmogorov-Smirnov set which makes the D_{KS} values discrete, may be the reason for rejecting the gaussianity.

However, if we limit ourselves to computing the normality test to random smaller subsets of the $\sqrt{D'_{KS}}$, it consistently accepts the null hypothesis of it being a normal, with p-values between 0.20 and 0.80.

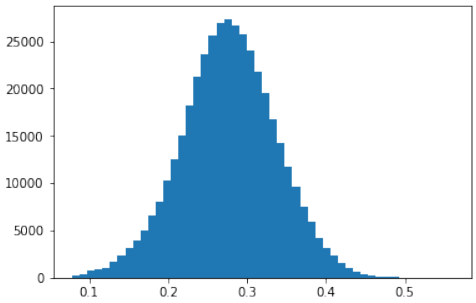
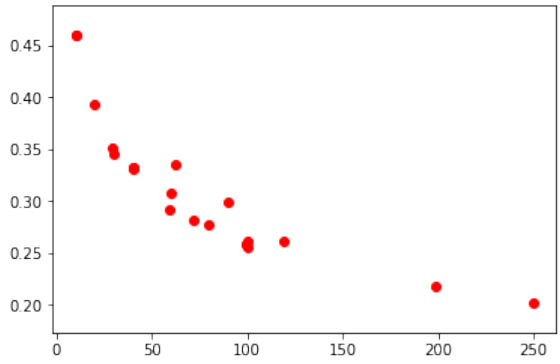
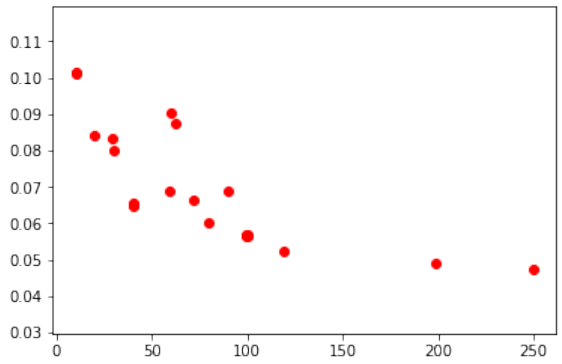


Figure 12: Histogram of *mit67TR* $\sqrt{D_{KS}(f, c)}$

Up until this moment, there seems to be no relationship between the D'_{KS} and the threshold predictability. We notice, though, that the parameters of this normal distribution can be related to it. With a high mean or high variance, the D'_{KS} values would be high and push the t thresholds away from 0. We therefore analyse these values for each of the datasets: in Figure 13 we plot these parameters with respect to the $\hat{I}_c$, which is the source of the predictability of our thresholds as seen in previous experiments. We observe that they arguably behave like the thresholds.



(a) Mean of the $\sqrt{D_{KS}(f,c)}$ distribution *w.r.t.* $\hat{I}_c$



(b) Standard deviation of the $\sqrt{D_{KS}(f,c)}$ distribution *w.r.t.* $\hat{I}_c$

Figure 13: Scatter plot of the parameters associated with the $\sqrt{D_{KS}(f,c)}$ *w.r.t.* target dataset's $\hat{I}_c$. Each point corresponds to a target dataset.

9.3 Studying FT threshold values

We have defined 5 thresholding methodologies: *Constant*, *Global*, *Feature-wise*, *Layer-wise*, *Feature-wise quantiles*. Before jumping to the accuracy results we first discuss results in the ft threshold resulting from these changes.

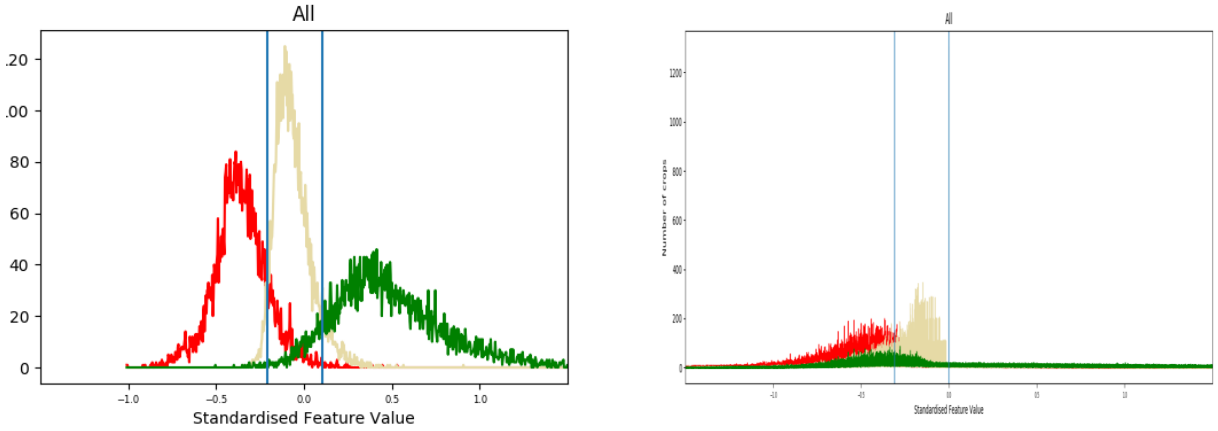
9.3.1 Studying behaviour of the defined methodologies

We start by studying the difference in behaviour of the methodologies when averaging the activations per group *w.r.t.* when not doing so.

Firstly, the *Global* methodology resulted in weird feature threshold behaviour. In Figure 14 we can observe the difference in behaviour between histograms of groupings for the same dataset. Figure 14a consists on the histograms taking the mean of each group per image, whereas Figure 14b considers all the activations. The later, contrary to what we would expect (histogram order red, grey, green, as in Figure 14a), shows discriminative groups (red and green) are partially superimposed, while the non-discriminative group (grey) is the one concentrated in a higher value. Note, however, that the positively discriminative (green) trails much further to the right than the rest.

This behaviour can be observed with a single stimulating image. In Figure 15 we show the distribution of activations for the three groups on an image of *mit67TR*. This behaviour shows that, even if a feature is positively discriminative toward a class, it may activate strongly for some of them and very weakly toward the rest, therefore exhibiting a behaviour similar to the negatively discriminative. This behaviour disappears somewhat when performing the mean of the activations, as the extremely positive feature activations raise the values of the histogram.

We also compare the mean against the no-mean behaviour when computing the thresholds *Layer-wise* in Figures 16 and 17. Remarkably, we notice the same differences that we saw in the *Global* methodology, but only in the deeper layers. We take particular notice on how in early layers of the non-mean *Layer-wise* methodology there seems to be a higher distinction between discriminative



(a) Mean of activations per group

(b) All activations per group

Figure 14: Feature activation histogram by grouping of feature *w.r.t.* class of stimulating image of *mit67TRTE* for *VGG16IN*.

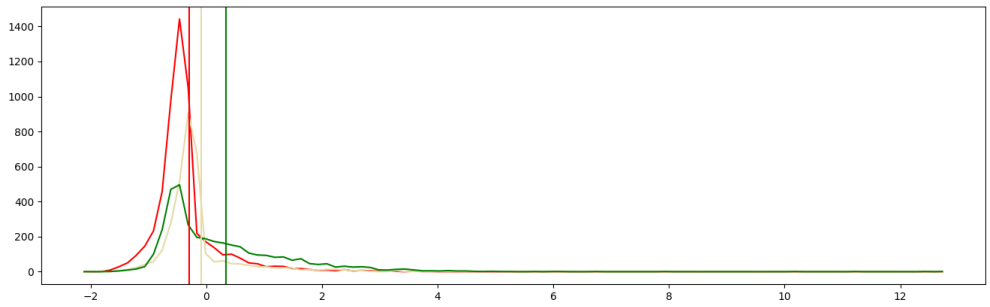


Figure 15: Histogram of activations for one image of *mit67TR*, with model *VGG16IN*

histograms. In both images we observe different behaviours of the distributions, but these are only pronounced in the last layers.

The erratic behaviours of the non-mean methodologies become problematic for obtaining thresholds. In addition, these behaviours also happen even when taking the mean in the *Feature-wise* methodology (given the smaller amount of activations considered). The most problematic behaviour was the feature threshold incoherence of $ft^+(f) < ft^-(f)$, which occurred frequently (one fifteenth of the times in *Feature-wise* methodologies), followed by the non-existence of one of the groupings (a feature never being non-discriminative, for example). Solving this problem is not trivial, and therefore we opted to remove non-mean methodologies from our analysis, and to patch the *Feature-wise* methodology with this mechanism:

If there one grouping does not exist, we take a single ft between the other two, and discretise to binary values $\{-1, +1\}$. We can be sure that there will always be at least two histograms.

If $ft^+ < ft^-$, we can take two possible solutions, with similar results. We can take a single threshold, the mean of the two thresholds (again discretising to two values). We can also interchange positive and negative thresholds, which still leaves three possible values for the output.

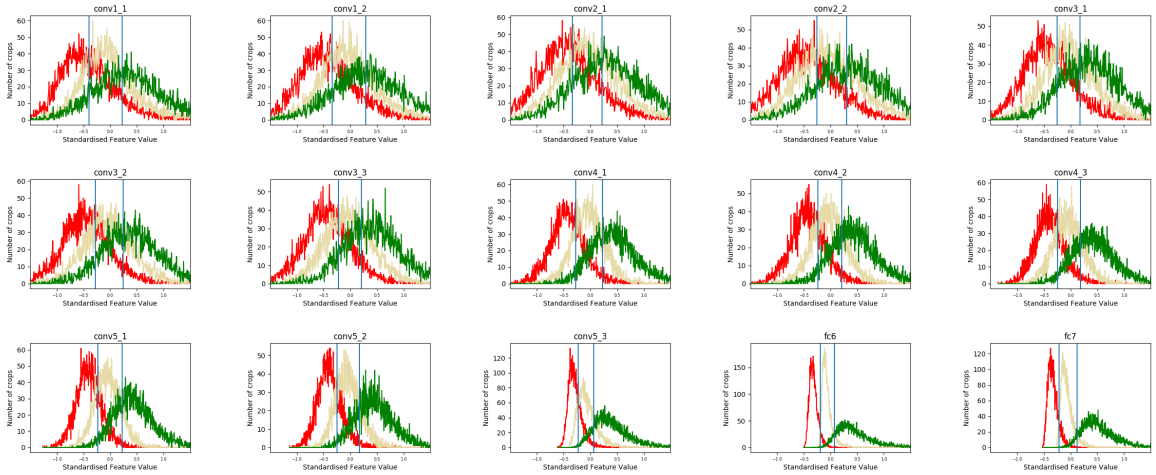


Figure 16: Feature activation histogram by grouping of feature *w.r.t.* class of stimulating image of *mit67TRTE* for *VGG16IN*, with layer granularity. Mean of activations per group

Both of these approaches resulted in accuracies, so we present them as a single methodology.

9.3.2 Method accuracy

We note once again that, even though *Feature-wise* has different alternatives for dealing with $ft^+ < ft^-$, the accuracies obtained were almost equal. For simplicity, only one entry has been submitted.

The datasets which used were *mit67*,*cub200*,*catsdogs*,*flowers102*,*textures*, for which we have both a train and a test set.

We expose the results in Figure 18, considering the normal accuracy and the balanced accuracy, which is the mean class accuracy (relevant given that our problems are imbalanced). Note that we are plotting the error of the accuracy $(1 - accuracy)$, as the small differences are more visible this way without adding possible bias from cropping. Therefore, for each target task, the best methodology is the shortest bar.

10 Discussion

10.1 Machine Learning strategies to deal with threshold acquisition

One of our initial hypothesis was that the standard deviation due to the stochasticity is small compared to the values obtained. This hypothesis seems validated by the results in Table 12, by the fact that the standard deviation is two orders of magnitude smaller than the mean threshold

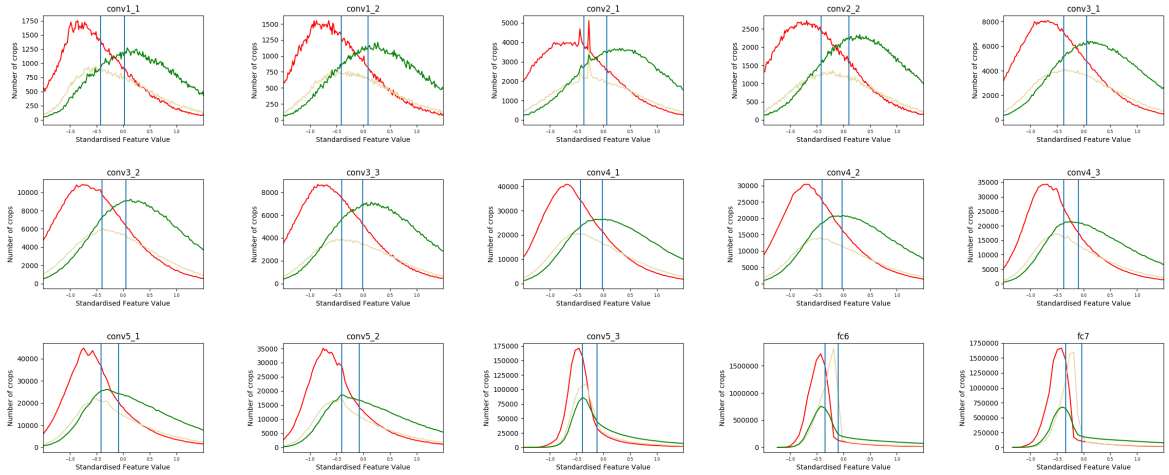


Figure 17: Feature activation histogram by grouping of feature *w.r.t.* class of stimulating image of *mit67TRTE* for *VGG16IN*, with layer granularity. All activations per group.

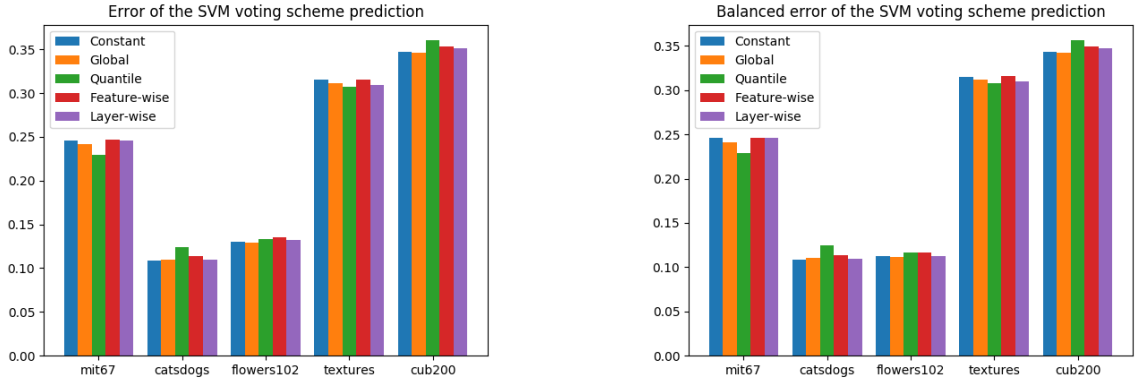
value. This same table also indicates the presence of an inverse correlation between the standard deviation for the thresholds and the $\hat{I}_c$ in the balanced tasks. To further validate this point, a more complete analysis would be needed.

Regarding the results from Table 13, we observe that the thresholds are highly predictable from I_c. We can attribute part of the error to neglecting class imbalance; notice how by removing the imbalanced tasks we drastically raise the R^2. This extraordinarily high predictability hints to the existence a mathematical relationship. Another significant finding is that the balanced regression (fitted with all balanced tasks) characterises better the *mit67TRTE* subsets' thresholds than the regression tailored for them. We attribute this to the sample size.

Comparing between pre-trained models (Figure 9) we find that regressions are almost superposed. We hypothesise the difference comes from a different discriminativity across the pre-trained models *w.r.t.* the targets. Even though it seems that a different topology (Figure 9b) yields a greater difference than different source task (Figure 9a), this is actually due to the outlier *caltech256TRTE*. If this task is removed, the difference is much less than that between the source tasks. The impact of both factors (source task and architecture) is thus minimal.

We find an outlier in the balanced regression (Figure 10a and Table 14): *food101TE*. While the task is balanced, the data point is the furthest away from the line, and has the second highest percentage of changes. We think this is caused by this target task being less discriminated against. Unlike other tasks where the average absolute D_{KS} is above 0.2, for this one is 0.15 (near the value of our predicted threshold). This means that feature-class pairs are not very discriminative. To optimise the threshold, the original method lowers the absolute value of the thresholds, raising the amount of noise but also of information. Since there are many feature-class pairs in this interval, small movements of the threshold heavily influence the amount of changes.

Regarding our analysis of the noise distribution, we discover that the parameters of the presumed



(a) Accuracy

(b) Balanced accuracy

Figure 18: Error of the metrics for each of the selected target tasks and source model *VGG16IN*.

gaussian that the square root of D'_{KS} follows seem to follow the same relationship as the previous regression. The mean and the standard deviation of the square roots of the D'_{KS} seem to be distributed according to a logarithmic reciprocal function from the $\hat{I}_c$ of the dataset that generated them. It seems that what we were predicting is actually the result of a process that is highly dependent on the parameters of this normal. The reason why the $\hat{I}_c$ influences the D'_{KS} in such a manner remains to be thoroughly analysed. We hypothesise that it might be related to the Wiener process inherent to the Brownian bridge followed by Kolmogorov-Smirnov:

$$K = \sup_{t \in (0,1)} B(t) = \sup_{t \in (0,1)} \left((1-t) W \left(\frac{t}{1-t} \right) \right)$$

In our case, the Brownian bridge associated is the motion of the distance between the two accumulated distributions, and each of the stochastic motion is associated to the activations. The process starts at 0, and increases or decreases every time an activation is reached. Whether this increases or decreases the value depends on whether the activation corresponds to inner or outer class. The Brownian bridge makes it so it starts and ends at 0, which is the behaviour expected from the distance of the two accumulated distributions. We hypothesise that the number of instances per class changes the probability of the stochastic movements of the Wiener process, resulting in the observed mean and standard deviation of the root of D'_{KS}.

Suppose a small dataset of ten categories and ten instances per class. When performing the D'_{KS} calculus, initially there is a 10% chance that the inner class distribution increases by 1/10, and a 90% chance that the outer class distribution increases by 1/90. This probabilities change depending on the previous consumed, but note that the expected value of the probabilities remains the same. In addition, an increase of I_c does not change the probabilities, but it decreases the increments associated to the accumulated distributions.

In summary: the instances per class drastically reduce the increments and decrements of the Brownian bridge (distances between the distribution), which, if we assume balanced classes, in turn reduces the maximum distance between the distribution, lowering the value of Kolmogorov-Smirnov. This, in turn, lowers the D'_{KS}, which lowers the thresholds t.

10.2 Studying FT threshold values

When studying the feature threshold behaviour we noticed that in no-mean methodologies the discriminative activations behave in similar manners. We might argue that the reason for this is related to the nature of the discriminativity itself.

A feature is considered positively discriminative if the accumulated distribution of the inner-class is significantly above the outer-class's. While being significant, this is also a simplification of the behaviour of the discriminativity. Consider a problem for classifying animals, and a class named horse. A feature is marked as discriminative for the class given its ability to recognise horse faces when they are the main focus of the picture, activating with very high strength. However, it activates weakly, even to the range of being negatively discriminative, when presented with the body of the horse. As a result, positively discriminative features can behave as negatively discriminative features even in front of the same class. However, since in the network there are many features, some of which may activate with higher strength in the case of the body pictures: this results in the right tail being very long in the non-mean methods, and in the positive histogram being pulled right in the mean methods.

This line of reasoning is coherent with the difference in layer behaviour of the *Layerwise* methodology. We observe that early convolutional layers behave differently from above, with the positive histogram being the right-most, converging toward the negative histogram as we increase the depth. We hypothesise that initial layers, whose convolutions tend to behave as Gabor filters, are discriminative for a bigger subset of the class. Suppose we have a task for classifying different categories, with each category having predominant textures. This texture could be associated to the fur and color of an animal or the background. However, the images in a category are presented from multiple perspectives, or pointing to different parts of the object to recognise. This fact produces later layer's behaviour, more focused on bigger objects, behave in specialised groups for each of the perspectives or parts, and activating weakly for the rest. However, the first layers are less specialised, and therefore regardless of the perspective they activate with strength when their predominant texture appears. This works particularly well when using textures as tells for when something is not of a class, which is the behaviour seen in the no-mean activations (Figure 17). This was proposed in previous work [7], and our results support their claims.

All of the five methodologies used yield comparable accuracies. We observe that the *Quantile* methodology seems to perform different from the rest, which is to be expected as it comes from a completely different approach. In some cases it outperforms the *Constant* by up to 1.5%, whereas in some cases it underperforms by the same amount, having the greatest variability when contrasted with the rest of methodologies.

In addition, the *Global* methodology seems to be slightly better than the *Constant* in all cases tested. Although this methodology requires more resources, we consider the optimisations done throughout this work will offset this issue and make it a viable solution.

The results coming from *Feature-wise* thresholds are somewhat surprising, but understandable. We hypothesised that by taking feature specific thresholds, each of the features' discretisation would keep the maximum amount of information, however they entail two problems. The first problem is the amount of information which we use to estimate these thresholds. Other methods use all the activations to compute the threshold, or subsets with a number of features. This one can only use the information of the concrete feature (only a vector of the size of the amount of

training images). This fact increases the stochasticity of the distributions, making noise more common, and possibly resulting in the aforementioned issues of $ft^+ < ft^-$. The second problem is that, by tailoring the thresholds to the training images to such a grain causes high overfitting of the data, to the point that it lowers the generalisation capabilities that the SVM can acquire. This is partially avoided by the *Quantile* methodology, which preserves a high amount of information in the data whilst avoiding to look at the information of the labels in the target task.

In the end, we consider that the best approach would be training the SVM with both the *Quantile* and the *Constant* methodologies to find the best one. If it is the latter, then substitute it with the *Global* for better accuracy.

11 Conclusions

Feature extraction for transfer learning has been studied in the past, from considering a single layer to the full network. Through these contributions we know that the representations learnt by a CNN can be used for new problems, which is of particular interest for new domains with no previous models trained in them.

By analysing and exploring the discriminativity of the individual features and classes, we begin to unveil characteristics and behaviours inherent to these inner representations. Particularly, we discover characteristics of the behaviour of this same discriminativity which allows us to remove computationally expensive sub-processes (Section 8.2.2,8.2.3). Moreover, these analysis are then used to dig up the activation behaviours of the features themselves in front of new stimuli, as well as neural behavioural difference between neurons both in the same layer and between layers (Section 8.2.4). This validates conclusions from previous work [7] and creates new ones, both of which we outline in this section.

From these we come with alternative methods of discretisation for the *Full-Network Embedding* defined in previous work [8], achieving similar accuracy in most cases, and finding plausible explanations for the cases which do not.

We outline the individual conclusions extracted from this work next:

1. The calculus of Kolmogorov-Smirnov between partitions of a distribution can be optimised to achieve up to 5 times with respect to library implementations when aggregating the histograms and subtracting instead of computing each partition individually (Section 8.2.1).

2. The stochasticity of label shuffling does not heavily modify the thresholds, as they may only produce slight modifications in the shuffled discriminativity distribution (Section 9.2.1).

3. The number of instances per class can be transformed into the final threshold with the formula 1, with low amount of error (Sections 9.2.1, 9.2.3).

4. Most of the error in the aforementioned formula seems to come from the class imbalance (Section 9.2.2).

5. We argue the remaining error might come from the difference between source and target tasks (Sections 9.2.2, 9.2.3).

6. The distribution of the shuffled discriminativity arguably follows a non-central chi-square, whose parameters are possibly a function of the number of instances per class and the imbalance (Section 8.2.3). We hypothesise this is a plausible reason why the threshold is predictable.

7. The origin of his distribution may come from the Brownian bridge behaviour of the Kolmogorov-Smirnov distance (Section 10.1).

8. We remark that global thresholding (setting the same two thresholds for all neurons), although empirically good, is dangerous as the activation distributions follow non-identical distributions even after standardisation (Section 8.2.4).

9. Results support the usefulness of early convolutional layers, specially when considering negative discriminativity (Section 9.3.1). Simultaneously, we also find that deeper layers have high specificity.

10. The behaviour of positively discriminative features is different across layers, and that deeper layer ones behave like the negatively discriminative ones in a majority of cases, activating with extraordinarily high values in a small amount of them (Section 9.3.1).

11. We propose a series of discretisation mechanisms for the standardised features, achieving comparable results. We note that the supervised and unsupervised mechanisms achieve similar accuracies (Section 9.3.2).

12. We remark the difference between the two unsupervised methodologies having the most dissimilar behaviour. One of them considers the behavioural difference between neurons and the other one does not (Section 9.3.2).

We consider that the main objectives of the project have been achieved, and in the process we have obtained valuable information of feature behaviour for transfer learning. However, we notice potential future work for the field, both related to the Full-Network Embedding and to other possible applications of transfer learning.

1. Regarding the function to predict discriminant thresholds, we might be able to reduce error from the imbalance the categories. This would be achieved through a more in-depth analysis and additions to the logarithmic reciprocal formula, and an analysis of the imbalance effect in the Brownian bridge of Kolmogorov-Smirnov.

2. We might be able to reduce the number of changed features by introducing information about the real discriminativity. This would be done by introducing a term in the logarithmic reciprocal formula, which slightly rises the threshold if the mean discriminativity is low (pushing the trade-off to get more discriminative features).

3. Results suggest that we should consider the positively discriminative features in deep layers as behaving both positively and negatively for a same category. Mean aggregation per image has proven useful to avoid this problem, but other methods (perhaps through different distance metrics) may obtain better results.

4. Results suggest that, as positively discriminative features present a dual behaviour in deeper layers, perhaps so do negatively discriminative ones. This may also factor in the previous analysis.

5. Discriminativity of the features offers possibilities regarding the removal of features of the embedding, as it is a novel approach for dimensionality reduction. This would cut training time for the machine learning method at the end of the pipeline.

6. Feature wise approaches for discretisation have proven to behave differently depending on whether they were supervised or unsupervised. Supervised methods behave poorly possibly due to overfitting of the data. Performing these methods with a reduced amount of data offers a possible supervised alternative which remains to be tested.

References

[1] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[2] D. Garcia-Gasulla and F. Parés, "Cnn slides from the deep learning course at mai." Available at `https://upc-mai-dl.github.io/mlp-convnets-theory/` (2019).

[3] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 806–813, 2014.

[4] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Advances in neural information processing systems*, pp. 3320–3328, 2014.

[5] H. Azizpour, A. S. Razavian, J. Sullivan, A. Maki, and S. Carlsson, "Factors of transferability for a generic convnet representation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 9, pp. 1790–1802, 2016.

[6] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *International conference on machine learning*, pp. 647–655, 2014.

[7] D. Garcia-Gasulla, F. Parés, A. Vilalta, J. Moreno, E. Ayguadé, J. Labarta, U. Cortés, and T. Suzumura, "On the behavior of convolutional nets for feature extraction," *Journal of Artificial Intelligence Research*, vol. 61, pp. 563–592, 2018.

[8] D. Garcia-Gasulla, A. Vilalta, F. Parés, E. Ayguadé, J. Labarta, U. Cortés, and T. Suzumura, "An out-of-the-box full-network embedding for convolutional neural networks," in *2018 IEEE International Conference on Big Knowledge (ICBK)*, pp. 168–175, IEEE, 2018.

[9] G. Hughes, "On the mean accuracy of statistical pattern recognizers," *IEEE transactions on information theory*, vol. 14, no. 1, pp. 55–63, 1968.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[12] B. Zhou, A. Khosla, A. Lapedriza, A. Torralba, and A. Oliva, "Places: An image database for deep scene understanding," *arXiv preprint arXiv:1610.02055*, 2016.

[13] A. Quattoni and A. Torralba, "Recognizing indoor scenes," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 413–420, 2009.

[14] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The caltech-ucsd birds-200-2011 dataset," 2011.

[15] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *2008 Sixth Indian Conference on Computer Vision, Graphics and Image Processing*, pp. 722–729, 2008.

[16] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. Jawahar, "Cats and dogs," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3498–3505, 2012.

[17] A. Khosla, N. Jayadevaprakash, B. Yao, and F.-F. Li, "Novel dataset for fine-grained image categorization: Stanford dogs," in *Proc. CVPR Workshop on Fine-Grained Visual Categorization (FGVC)*, vol. 2, 2011.

[18] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," *Computer vision and Image understanding*, vol. 106, no. 1, pp. 59–70, 2007.

[19] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.

[20] L. Bossard, M. Guillaumin, and L. Van Gool, "Food-101–mining discriminative components with random forests," in *European Conference on Computer Vision*, pp. 446–461, Springer, 2014.

[21] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, "Describing textures in the wild," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3606–3613, 2014.

[22] O. Silvén, M. Niskanen, and H. Kauppinen, "Wood inspection with non-supervised clustering," *Machine Vision and Applications*, vol. 13, no. 5, pp. 275–285, 2003.